部分語計数問題の接尾辞配列を用いた高速アルゴリズム (計算モデルとアルゴリズム)

AUTHOR(S):

笠井, 透; 有村, 博紀; 有川, 節夫

数理解析研究所講究録
1093巻 1999年 81-86

部分語計数問題の接尾辞配列を用いた高速アルゴリズム

笠井 透　　有村博紀　　有川節夫

九州大学大学院システム情報科学研究科 情報理学専攻

1 はじめに

大規模テキストデータベースの急速な発展によって, テキストデータから規則性やパタンを発見する研究が注目されている. 従来のパタン発見や文字列解析のアルゴリズムの多くは, 接尾辞木 (suffix tree)[8] とよばれる索引構造を対象としている. 最近, よりコンパクトなデータ構造である接尾辞配列 (suffix array)[9] が提案され, 大規模テキストデータベースにおけるデータ構造として注目されている. この接尾辞配列は, 実現において接尾辞木の $1/2 \sim 1/3$ の記憶容量しか使用しない.

本研究では, 大規模テキストデータベースからの効率よいパタン発見を実現するために, 接尾辞配列上で高速なパタン発見を可能にするための基本的実装法について考察する. 接尾辞配列を左から右へ一度走査するだけで接尾辞木の仮想的巡回をおこない, テキスト中のすべての部分語の頻度を計算する高速なアルゴリズムを提案する. このアルゴリズムの時間計算量は $O(n)$ であり, 2分探索を繰り返し用いて木の巡回を模倣する素朴なアルゴリズムの $O(n \log n) \sim O(n^2)$ に比べるとオーバーヘッドが小さい. したがって, パタン探索問題やテキストデータマイニングの高速化に有効である. また, 計算機実験の結果も示す.

2 準備

2.1 接尾辞木

本稿では, 記号のアルファベット Σ に対して, Σ 上の任意の文字列 $s \in \Sigma^*$ を**語** (word) とよび, その長さを $len(s)$ で表す. 語 s に対して, $s = uvw$ を満たす語 u, v, w を, それぞれ, s の**接頭辞** (prefix), **部分語** (subword), **接尾辞** (suffix) とよぶ. 語 s, t に対して, s と t に共通する最長の接頭辞を**最長共通接頭辞** (longest common prefix) といい, その長さを $lcp(s,t)$ で表す.

長さ n のテキスト (text) とは, 文字列 $A = a_1 a_2 \cdots a_{n-1}\$$である. ここに, $a_i \in \Sigma$ であり, 記号 \$ は\$ $\notin \Sigma$ であるような特別な区切り記号である. 本稿では, $n \geq 2$ と常に仮定する. 語 s が, ある整数 $1 \leq i, j \leq n, i \leq j$ に対して, $s = a_i \cdots a_j$となるとき, s は A に**出現する**といい, i を s の**出現位置** (occurrence) という. 任意の $1 \leq i \leq n$ に対して, 位置 i からはじまる A の接尾辞を $A_i = a_i \cdots a_{n-1}\$$ で表す.

テキスト A の**接尾辞木** (suffix tree) とは, つぎのように定義される順序木 T_A である [8]:

(1) 各辺は, A の空でない部分語α をラベルとしてもつ. ラベル α は, その出現位置 i と終了位置 $i+len(\alpha)-1$ の組 $(i, i+len(\alpha)-1)$ で符号化されている.

(2) 任意の内部節点に対して, その子へと出ていくすべての辺のラベルは, 先頭の文字が互いに異なる.

(3) 各節点 v は, 根から v へ至る辺のラベルを合併して得られる語を表す. これを, **分岐語** (branching subword) とよび, $Word(v)$ と書く.

(4) 葉を n 個もち, それらの葉が表す分岐語は, A の空でない接尾辞である. 各葉は, それが表す接尾辞の A 中の開始位置をもち, A の空でないすべての接尾辞が, 左の葉から右の葉へ$\Sigma \cup \{\$\}$ 上の辞書式順序で並んでいる.

図1の左に, 接尾辞木の例を示す.

接尾辞木 T_Aは, n 個の葉とたかだか $n-1$ 個の内部節点をもち, $O(n)$ 領域を使う. 整数を4バイトで表現すると, T_Aは $15n$ バイトの記憶領域を必要とする. McCreight [8] は, T_A を計算する $O(n)$ 時間アルゴリズムを与えている.

Virtual suffix trees: fast computation of subword frequency using suffix arrays, T. Kasai, H. Arimura, S. Arikawa, Department of Informatics, Kyushu University, Hakozaki 6-10-1, Fukuoka 812-8581, Japan. {arim, arikawa}@i.kyushu-u.ac.jp

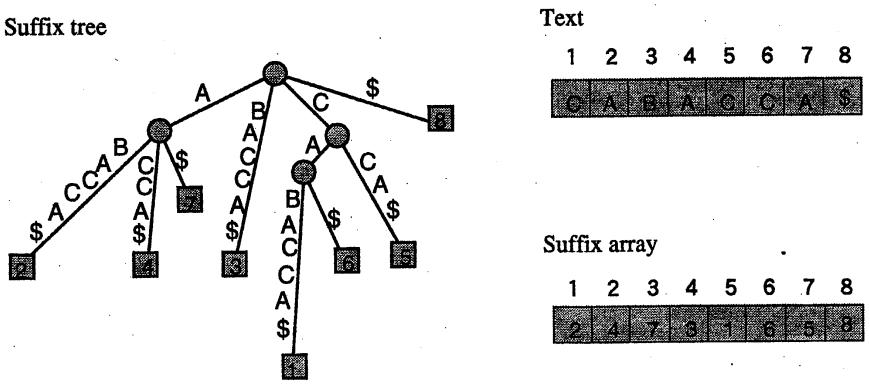

図 1: 接尾辞木と接尾辞配列

2.2 接尾辞配列

テキスト A の**接尾辞配列** (suffix array) は,A の各接尾辞へのポインタを格納した 1 次元配列 $Pos[1..n]$ であり, 各ポインタは, それが指し示す接尾辞の辞書式順序でソートされている. 定義より, 任意の $1 \leq i \leq n$ に対して, $Pos[i]$ は, 辞書式順序で順位が i の接尾辞の A 中での開始位置である. これは, 接尾辞木の葉だけを左から右に一次元整数配列に格納したものに等しい.

図 1 の右側下に, 接尾辞配列の例を示す. 例えば, 左から 2 番目のセルの値 4 は, テキストの 4 文字目からはじまる順位 2 の接尾辞 $A_4 = \texttt{ACCA\$}$ を表す. また, 長さ 1 の文字列 A の出現位置は, 接尾辞配列の連続したセル [1..3] を占めることもわかる.

$Pos[1..n]$ に対して, その**逆関数となる配列** $Suf[1..n]$ を, $Suf[Pos[i]] = i$ と定義する. $Suf[i]$ は, 接尾辞 A_i の順位である. **高さ配列** (hight array)$Hgt[i]$ は, 任意の順位 $1 \leq i < n$ に対して, $Hgt[i] = lcp(A_{Pos[i]}, A_{Pos[i+1]})$ で定義される配列である. ただし, $Hgt[n] = -1$ と定義する.

接尾辞配列 Pos は, テキストとあわせて $5n$ バイトしか使わない. Pos は, いったん接尾辞木を計算することで $O(n)$ 時間で構成できるが, 実用的には, クイックソートを用いた平均時間 $O(n \log n)$ のアルゴリズムを用いることが多い [4].

2.3 順序木と巡回

順序木 T は, そのすべての内部節点が 2 つ以上の子をもつとき, **コンパクト**であるという. 接尾語木は, コンパクトな順序木である. 木 T の節点を u, v, w とおく. 節点 u, v に対して, u が v の**先祖**であることを, $u \preceq v$ で表し, **真の先祖**であることを $u \prec v$ で表す. T の葉 l が, 左から i 番目の葉であるならば, l の順位は i であるという. 節点 v に対して, $left(v)$ ($right(v)$) を v を根とする T の部分木の最左の葉の順位 (最右の葉の順位) と定義する.

節点 w が, u と v の**最近共通先祖** (nearest common ancestor) とは, 節点 $w = nca(u, v)$ で $w \preceq u$ かつ $w \preceq v$ であり, $x \preceq u$ かつ $x \preceq v$ とするすべての節点 x に対して, $x \preceq w$ が成立するものをいう.

リストの連結を $\cdot$ で表す. T の**後置順巡回** (postorder traversal) とは, つぎのように再帰的に定義される節点のリストである:

(1) T がただ一つの節点 v からなるとき, (v) は, T の後置順巡回である.

(2) T の根 v が, 子 $v_1, v_2, \ldots, v_m$ $(m \geq 1)$ をもつとし, 各 i に対し, v_i を根とする T の部分木の後置順巡回を Γ_i とすると, $\Gamma_1 \cdot \Gamma_2 \cdot \ldots \cdot \Gamma_m \cdot (v)$ は, T の後置順巡回である.

2.4 パタン探索問題

パタン発見に関わる問題の多くは, 各節点の巡回と動的計画法の適用によって効率よく計算できる. このクラスの問題を一般化する.

D を値の集合とし, これを**領域** (domain) とよぶ. 演算子 $\oplus : D \times D \to D$ を, 空要素 ϕ をもつ D 上の結合的二項演算子とする. ただし, ϕ と任意の値 $e \in D$ に対して, $e \oplus \phi = \phi \oplus e = e$ とする. **初期割り当て** $B : \{1, \ldots, n\} \to D$ を, テキスト A 上の任意の位置への値の割り当てとする. テキスト A の**部分語統計** (subword statistics) とは, A の任意の部分語 α から領域 D への写像 C であり, A 中での α の出現位置全体 $i_1 \leq i_2 \leq \cdots \leq i_m$ に対して, $C(\alpha) = B(i_1) \oplus \cdots \oplus B(i_m)$ と定義する.

Naive_Traverse
1. T_A の任意の葉 l_i に, $C_{l_i} := B(p)$ を割り当てる. ここに, p は, l_i が表す接尾辞の A 中の開始位置とする.
2. 葉から根へと走査しながら, 各内部節点 v に対して, リスト $C_v := C_{v_1} \oplus C_{v_2} \oplus \cdots \oplus C_{v_m}$ を対応づける. ここに, $m \geq 2$ であり, 任意の $1 \leq i \leq m$ に対して, v_i は, v のもつ左から i 番目の子とする.

図 2: テキストのすべての部分語の部分語統計を, 接尾辞木の深さ優先探索を用いて計算するアルゴリズム

部分語統計問題 (subword statistics problem)
入力: テキスト A, 領域 D, 初期の割り当て B, 二項演算子$\oplus$.
問題: A の (出現位置がことなる) すべての部分語 α に対して, 部分語統計 $C(\alpha)$ を出力せよ.

部分語統計問題は, 接尾辞木の深さ優先探索を用いて, 図 2 のように計算できる.

以下の問題は, 上記の部分語統計問題の具体例またはそれに密接に関連した問題の例である:

- 部分語頻度計算問題 (String statistics with overlaps) [1]. 入力テキストの (出現位置がことなる) すべての部分語について, その出現回数を答えよ.
- 最長反復部分語問題 (Longest repeated substring problem) [3] (pp. 21). 入力テキスト中に 2 回以上出現する最長の文字列をみつけよ.
- 最長共通部分語問題 (Longest common substring problem) [3] (pp. 20). 2 つの入力テキスト中に共通して出現する最長の文字列をみつけよ. (一般には, 入力テキストの数が任意の場合も考えられる.)
- 出現文書数問題 (Color set size ploblem) [6]. 入力テキストの集合が与えられたとき, (出現位置がことなる) すべての部分語について, それが出現する文書数 (color set size) を答えよ.
- 部分語無矛盾性問題 (Characteristic substring problem) [10]. 入力として, 2 つのテキスト集合 P, N が与えられたとき, P 中のすべてのテキストに出現し, N 中のどのテキストにも出現しない文字列を見つけよ.

2.5 素朴な模倣アルゴリズム

接尾辞配列をもちいて部分語統計問題を解く方法として, 接尾辞木の深さ優先探索をおこなう図 2 のアルゴリズムを, そのまま接尾語配列上で模倣することが考えられる.

このとき, 各節点 v を, それが占める順位のなす区間 $(left(v), right(v))$ で表現する. さらに, 節点での分岐を接尾辞配列上での 2 分探索で模倣し, スタックを用いて深さ優先探索をおこなう. しかしこの手法の計算時間は $O(n \log n + Q + M(n))$ 時間となる. ここに, Q は非圧縮接尾辞トライ $\bar{T}_A$ の節点数 $Q = O(n^2)$ であり, $M(n)$ は, 図 2 のアルゴリズムにおける$\oplus$演算の所要時間の合計である. 次節以降では, より効率よく接尾辞木の巡回を模倣する方法を与える.

3 コンパクト順序木のボトムアップ巡回

本節と次節では, 接尾辞配列上で接尾辞木の巡回を模倣しながら, テキストに現れるすべての部分語の部分語統計を計算する効率よいアルゴリズムを与える.

まず準備として, 本節では, 一般の順序木において, 葉の左から右への走査と, 最近共通節点の計算, 節点間の先祖関係の計算だけが基本的演算として与えられている場合に, 木の巡回をボトムアップにおこなうアルゴリズムを与える. 次に次節では, このアルゴリズムを接尾辞配列上で実現可能なことを示す. これにより, 接尾辞木の巡回を接尾辞配列を用いて効率よく実現するアルゴリズムを与える.

3.1 アルゴリズム

図 3 に, 接尾辞木の葉を左から右へ走査することで, 後置順巡回で各節点を巡回するアルゴリズムを示す. アルゴリズムは, スタック S を用いて木を巡回する.

本節では, T を, n 個の葉をもつコンパクトな順序木とする. 任意の $1 \leq i \leq n$ に対して, l_i で T の左から i 番目の子を表し, nca_i で葉 l_i と l_{i+1} の最近共通先祖 $= nca(l_i, l_{i+1})$ を表す. ただし, 特別な節点$\perp$を用いて, $nca_0 = nca_n = nca(l_0, l_1) = nca(l_n, l_{n+1}) = \perp$と定義する. 根の仮想的な親が$\perp$である.

3.2 正当性

定義 1 (**最長最右枝**) 順序木 T の i 番目の葉 l_i に対して, 葉 l_i から根へ進む経路で, 節点の最右辺だけから構成される最長のものを, **最長最右枝**といい, Π_iと書く. ここに, Π_i は節点のリストとして表現されているとする.

任意の順序木は, 最長最右枝の集まりとして表現できる. 図 4 に, 順序木の最長最右枝の例を示す.

Algorithm *Traverse_Tree*
1. Compute the nca_i s and $S := \phi$.
   Push $\hat{r}$ into S.
2. For each leaf l_i, $i = 1, \ldots, n$, do:
(a) Push l_i.
(b) $w := nca_i$. Let v be the top of S.
(c) While $w \prec v$, do:
   (i) Report v.
   (ii) Let v be the top of S.
(d) If $v = w$ then
      Do nothing.
(e) Else if $v \prec w$ then
      Push w into S.

図 3: 順序木の葉を左から右へ走査しながら, 後置順巡回で各節点を巡回するアルゴリズム

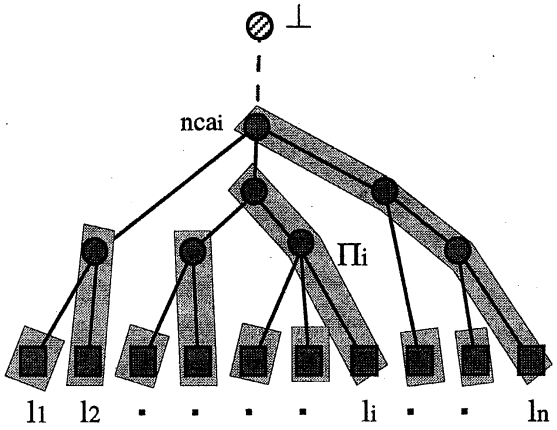

図 4: 順序木の各葉 l_i の最長最右枝. 各葉から根へ進む灰色の経路が最長最右枝である.

補題 1 任意の $1 \leq i \leq n$ に対して, 葉 l_iから 節点 nca_iへの経路を $v_1(= l_i)v_2 \cdots v_k v_{k+1}(= nca_i)$ とおく. このとき, $\Pi_i = v_1 v_2 \cdots v_k$ である.

定理 2 葉数 $n \geq 1$ のコンパクトな順序木を T とおく. このとき, T のすべての最長最右枝を左から右へ連結して得られるリスト $\Pi_1 \cdot \Pi_2 \cdot \ldots \cdot \Pi_n$ は, T の後置順巡回に等しい.

アルゴリズム *Traverse_Tree* において, ステップ 2.(a) からステップ 2.(e) の For ループの i 回目の実行を**第 i ステージ**とよぶ. 第 i ステージにおいて, ステップ 2.(a) でスタックへのプッシュを行った直後のスタックの内容を S_{i-1}とおく. スタックの内容は, $S = v_k v_{k-1} \cdots v_1 \perp$のように, スタックの頂上を左に, 底を右に向けた要素の列として表す.

補題 3 アルゴリズムの第 i 番目のステージのステップ 2.(a) でスタックへのプッシュを行った直後において, スタックの内容 $S_{i-1} = v_{top} v_{top-1} \ldots v_1 \perp$ は, つぎの (1) と (2) を満たす:

(1) スタックの任意の要素を v_j $(0 \leq j < top)$ とする. 一つ上の要素 v_{j+1} を根とする部分木を考え, その最左の葉を l_kとおく. このとき, $v_j = nca(l_{k-1}, l_k)$ が成立する.

(2) スタック中の節点は, 葉 l_i から根にいたる経路上に順に並んでいる. すなわち, $v_0 = \perp \leq_T v_1 \leq_T \cdots \leq_T v_{top} = l_i$ である.

補題 4 コンパクトな順序木を T とし, その任意の節点を v とする. 節点 v を根とする T の部分木の最左の葉を l_iとし, 最右の葉を l_jとする $(i \leq j)$. さらに, l_iの一つ左隣の葉を l_{i-1}とし, l_jの一つ右隣の葉を l_{j+1}とする. このとき, このとき, v の親は, $nca(l_{i-1}, l_i) = nca_{i-1}$ と $nca(l_j, l_{j+1}) = nca_j$ のどちらかに等しい.

補題 5 アルゴリズムの第 i 回目のステージにおいて, ステップ 2.(a) の実行直後には, 最長最右枝 Π_i が, スタックの最上部に積まれている. すなわち, スタック $S_{i-1} = v_{top} \cdots v_1 \perp$ に対して, $\Pi_i = v_{top} \cdots v_k$ となる整数 $0 \leq k \leq top$ が存在する.

定理 6 コンパクトな順序木 T が与えられたとき, 図 3 のアルゴリズム *Traverse_Tree* は, T のすべての節点を後置順巡回で巡回する.

4 接尾辞配列による高速な巡回

本節では, 前節のアルゴリズムを用いて, 接尾辞木の巡回を接尾辞配列を用いて効率よく実現する線形時間アルゴリズムを与える. さらに, これを用いて, 接尾辞配列からテキストに現れるすべての部分語の部分語統計を計算する効率よいアルゴリズムを与える.

この節を通して, 長さ n のテキストを A とし, A の接尾辞木を T_Aとおく.

4.1 アルゴリズム

図 5 に, 前節のアルゴリズム *Traverse_Tree* を接尾辞配列上で模倣するアルゴリズム *Traverse_with_Array* を示す. アルゴリズムは, 接尾

Algorithm *Traverse_with_Array*
1. Compute $Hgt[1..n]$ and $S := \phi$. Push $(\phi, (0, -1))$ into S.
2. For each rank $i = 1, \ldots, n$, do:
(a) Push $(B(Pos[i])), (i, |A_{Pos[i]}|))$.
(b) $(C_{new}, (L_{new}, H_{new})) := (\phi, (i, Hgt[i]))$. Let $(C, (L, H))$ be the top of S.
(c) While $H > H_{new}$, do:
(i) Report $(C \oplus C_{new}, (L, H))$.
(ii) $C_{new} := C \oplus C_{new}$ and pop S. Let $(C, (L, H))$ be the top of S.
(d) If $H = H_{new}$ then
Pop $(C, (L, H))$ from S, and then push $(C \oplus C_{new}, (L, H))$ into S.
(e) Else if $H < H_{new}$ then
Push $(C_{new}, (L_{new}, H_{new}))$ into S.

図 5: 接尾辞配列をもちいて, 部分語統計問題を解くアルゴリズム

辞配列を左から右へ走査しながら木の巡回を模倣し, テキストに現れるすべての部分語の部分語統計を計算する.

4.2 正当性

テキスト A の接尾辞木を T とする. 任意の整数の組 (L, H) に対して, つぎの (1) と (2) をみたす節点 v が存在するとき, $node(L, H) = v$ と定義する:

(1) $left(v) \leq L \leq right(v)$.

(2) $len(word(v)) = H$.

もし条件をみたす v が存在しないなら, $node(L, H)$ は未定義とする.

この定義から $node(L, H)$ が唯一に定まることが容易にわかる. アルゴリズムでは, 節点 v を $node(L, H) = v$ となるような組 (L, H) で表す.

補題 7 任意の整数 $1 \leq i \leq n$ に対して, $node(i, len(A_{Pos[i]})) = l_i$が成立する.

補題 8 任意の整数 $1 \leq i \leq n$ に対して, $node(i, Hgt[i]) = nca_i$が成立する.

補題 9 アルゴリズムの第 i 回目のステージのステップ 2.(c)–ステップ 2.(e) において, スタックの頂上の要素 $(L, H) = top(S)$ と $(L_{new}, H_{new}) = (i, Hgt[i])$ に対して, つぎの (1)–(3) が成立する.

(1) $H > Hgt[i] \iff node(L, H) \succ nca_i$.

(2) $H = Hgt[i] \iff node(L, H) = nca_i$.

(3) $H < Hgt[i] \iff node(L, H) \prec nca_i$.

定理 10 長さ n のテキスト A および A の接尾辞配列 Pos が与えられたと仮定する. このとき, 図 3 のアルゴリズム *Traverse_with_Array* は, $O(n)$ 時間で, T_Aのすべての節点を後置順巡回で訪問する.

Proof: 補題 6 および, 補題 7, 補題 8, 補題 9 から導かれる. □

系 11 長さ n のテキスト A および, A の接尾辞配列 Pos, 部分語統計問題 $(D, B, \oplus)$ が与えられたとする. このとき, 図 3 のアルゴリズム *Traverse_with_Array* は, テキスト A 中のすべての部分語の部分語統計を $O(n + M(n))$ 時間で計算する. ここに, $M(n)$ は, $\oplus$ 演算の所要時間の総計である.

Proof: 定理 10 から, アルゴリズム *Traverse_with_Array* は, T_Aの後置順巡回を正しく模倣する. 後置順巡回では, ある節点 v が訪問されるとき, その子どもはすでに訪問されており, v に関連づけられた値 C_vはすでに計算ずみであることが保証される. よってアルゴリズムは, すべての節点 v に対して正しく $C(word(v))$ を出力する. 計算時間については, Push と Pop 演算は $O(1)$ 時間で実行でき, $len(A_{Pos[i]}) = n - i + 1$ なのでこれも $O(1)$ 時間で計算可能である. また, $\oplus$演算の実行回数は, 図 2 のアルゴリズム *Naive_Traverse* でのものと変わらない. よって, 構成より明らか. □

5 高さ配列 Hgt の線形時間計算

前節のアルゴリズムでは, 高さ配列 Hgt を用いて接尾辞木の仮想的な巡回をおこなった. 本節では, テキスト配列と接尾辞配列から Hgt を線形時間で計算するアルゴリズムを与える.

定義から, Hgt はすべての $1 \leq i \leq n$ に対して, $Hgt[i] = lcp(A_{Pos[i]}, A_{Pos[i+1]})$ を計算することで求められる. しかし, 一般に $lcp(A_{Pos[i]}, A_{Pos[i+1]}) = O(n)$ であるので, この簡単な方法では最悪時に $O(n^2)$ 時間を要する[1].

図 6 に, 高さ配列 $Hgt[1..n]$ を $O(n)$ 時間で計算するアルゴリズム *Fast_Hgt* を示す. 次の補題は, アルゴリズムの正当性に本質的である.

補題 12 任意のテキスト A と整数 $1 \leq i < n$ に対して, $lcp(A_{Pos[i]}, A_{Pos[i+1]}) - 1 \leq lcp(A_{Pos[i]+1}, A_{Pos[i+1]+1})$ が成立する.

[1] 例えば, $A = aaaaaa \cdots a$ の場合.

Algorithm *Fast_Hgt*
1. Compute $Suf[1..n]$ and $h := 0$.
2. For each position $i = 1, 2, \ldots, n$, do:
   (a) If $Suf[i] = n$ then
   $Hgt[Suf[i]] = -1$ and continue.
   (b) $j := Pos[Suf[i] + 1]$.
   (d) If $h = 0$ then
   $Hgt[Suf[i]] := lcp(A_i, A_j)$.
   (c) Else if $h > 0$ then
   $Hgt[Suf[i]] := h-1+lcp(A_{i+h-1}, A_{j+h-1})$.
   (e) $h := Hgt[Suf[i]]$.

図 6: 高さ配列 $Hgt[1..n]$ の線形時間アルゴリズム

Proof: テキスト A において, $A_{Pos[i]}$の先頭から 1 文字取り除いたものが $A_{Pos[i]+1}$である. 同様に, $A_{Pos[i+1]}$の先頭から 1 文字取り除いたものが $A_{Pos[i+1]+1}$である. よって, $A_{Pos[i]}, A_{Pos[i+1]}$の共通接頭辞の先頭から 1 文字取り除いたものは, $A_{Pos[i]+1}, A_{Pos[i+1]+1}$の共通接頭辞になる. このことから導かれる. □

アルゴリズム *Fast_Hgt* は, テキストを左から右へ走査し, 位置 $Pos[i]$ を増加させながら, $lcp(A_{Pos[i]}, A_{Pos[i+1]})$ を計算していく. 上の補題から, アルゴリズムは, 接尾辞同士の重複部分を利用して文字列比較の回数を減らし, Hgt を高速に計算する.

補題 13 長さ n のテキスト A と A の接尾辞配列 $Pos[1..n]$ が与えられたとき, 図 6 のアルゴリズム *Fast_Hgt* は, $Hgt[1..n]$ を $O(n)$ 時間で計算する.

6 計算機実験

接尾語木の巡回を 2 分探索によって模倣する素朴なアルゴリズムと今回提案するアルゴリズム Fast_Traverse (図 7) を, Unix ワークステーション (Sun Enterprise3000, g++ on Solaris 2.5) 上に実現し, 5.3MB の英文テキスト [5] を対象として計算時間を測定した. 接尾辞配列は主記憶上においた.

次ページの図 7 に計算機実験の結果を示す. 上の 2 つの欄は, 素朴な巡回アルゴリズム *Binary_Traverse* と提案の巡回アルゴリズム *Fast_Traverse* について, 巡回の時間を示す (前処理で Hgt を計算する時間は含まない). 下の 2 つの欄は, 前処理における Hgt 配列の計算について, 素朴なアルゴリズム *Naive_Hgt* と提案のアルゴリズム *Fast_Hgt* の計算時間を示す.

使用領域は, Pos と Hgt がそれぞれ $4n$ と $2n$ バイトであり, スタック平均長は n よりかなり小さい. また, アルゴリズムの単純さも長所である.

Algorithm	Binary_Traverse	Fast_Traverse
Time (sec)	13.62	1.94
Algorithm	Naive_Hgt	Fast_Hgt
Time (sec)	17.59	7.81

図 7: 計算時間の比較

本アルゴリズムはディスク上での実装にも適する. それには, 前処理で計算した Hgt と Pos をディスクにおき, スタックを主記憶上におけばよい. この際, Hgt のアクセスパタンは逐次的である.

7 おわりに

本稿では, 高速なパタン探索手法について論じ, 接尾辞配列上で接尾辞木の巡回を実現する線形時間アルゴリズムを与えた. ここで略したアルゴリズムと証明の詳細については [7] を参照されたい.

本手法を用いて, 文献 [2] のテキストデータマイニングにおける語相関パタン発見問題が, 接尾辞木を用いた場合と同じ時間計算量で実装可能である. 詳細に関しては, 別の機会に述べたい.

参考文献

[1] A. Apostolico, F. P. Preparata, Structural properties of the string statistics problem. *JCSS*, 31(3):394-411 (1985)

[2] H. Arimura, S. Shimozono, Maximizing agreement between a classification and bounded or unbounded number of associated words. In *Proc. ISSAC*, (1998).

[3] M. Crochemore, W. Rytter, Text Algorithm. *Oxford University ress* (1994)

[4] G. H. Gonnet, R. Baeza-yates, T. Snider, New indices for text: Pat tree and Pat arrays. Information Retrieval, Prentice Hall (1992).

[5] R. Harris, Abstract Index, Monash Univ. (1998).

[6] L. C. K. Hui, Color set size problem with applications to string matching. In *Proc. of 3rd CPM*, 230-243 (1992).

[7] 笠井 透, 部分語計数問題の接尾辞配列を用いた高速アルゴリズム. 修士論文, 九州大学大学院システム情報科学研究科情報理学専攻, 平成 11 年 2 月.

[8] E. M. McCreight, A space-economical suffix tree construction algorithm. *JACM*, 23(2):262-272 (1976).

[9] U. Manber, G. Myers, Suffix arrays: A new method for on-line string searches. *SIAM J. Computing*, 22(5):935-948 (1993).

[10] M. Nakanishi, M. Hashidume, M. Ito, A. Hashimoto, A linear-time algorithm for computing characteristic strings. In Proc. 5th ISAAC (1994), 315-323.